

## Triggers and Functions

### Triggers

DROP TRIGGER statement:

Infx: DROP TRIGGER trigname;

psql: DROP TRIGGER trigname ON tablename;

CREATE TRIGGER statement:

Infx: CREATE TRIGGER trigname  
      INSERT ON tablename  
      REFERENCING NEW AS xxxx  
      FOR EACH RIW  
      (EXECUTE PROCEDURE procname (arg1, arg2, ...));;

psql: CREATE TRIGGER trigname  
      BEFORE INSERT ON tablename

see Douglas pp 302 - 305

### Functions

called Stored Procedures in Infx

stored in and executed by server

functions called by triggers must be written in one of the procedural languages provided by psql

PL/pgSQL language is the postgresql procedural language  
- "shares a striking resemblance to Oracle ..."

PL/pgSQL does not have ON EXCEPTION statement

- it has a RAISE EXCEPTION statement which will write a message to a log, terminate function and abort current transaction
- "error handling is PL/pgSQL's weak point ..." see Douglas p 289
- there is no "EXEC SQL EXECUTE PROCEDURE" statement in psql
  - SELECT function\_name(); can be used to execute a function

PL/Tcl, PL/Perl and PL/Python allow creation of functions using a subset of the language - psql does not allow I/O external to the db

language must be installed at command line using createlang command  
for example:

```
createlang plpgsql hd_ob6rha -U postgres
```

in shefdecode, Stored Procedures are called via the execProcedure function

- code has "EXEC SQL EXECUTE PROCEDURE ..." statement
- statement does not exist in psql
- replaced execProcedure function with execFunction for psql
- "EXEC SQL EXECUTE PROCEDURE ..." replaced by "EXEC SQL SELECT ..."

\*\*\*\*\*

Implemented insert trigger/function on Precip table to insert record into CurPrecip table

Function looks as follows:

```
CREATE OR REPLACE FUNCTION obs_precip_ins () RETURNS TRIGGER AS '
BEGIN

    INSERT INTO CurPrecip
    VALUES
    (
        NEW.lid,
        NEW.pe,
        NEW.dur,
        NEW.ts,
        NEW.extremum,
        NEW.obstime,
        NEW.value,
        NEW.shef_qual_code,
        NEW.quality_code,
        NEW.revision,
        NEW.product_id,
        NEW.producttime,
        NEW.postingtime
    );
    RETURN NEW;
END;

' LANGUAGE 'plpgsql';
```

Trigger statement looks as follows:

```
create trigger obs_precip_ins
    after insert on precip
    for each row
        execute procedure obs_precip_ins ();
```

\*\*\*\*\*  
Implemented delete\_radar function in PostgreSQL as follows:

```
-- -- Supports a cascading delete when
-- a row is removed from the RadarLoc table.
--
```

```
CREATE OR REPLACE FUNCTION delete_radar (char(3)) RETURNS INT AS '
DECLARE delradid ALIAS FOR $1;
BEGIN
    delete from DpaAdapt      where radid = delradid;
```

```

    delete from    RW RadarResult      where radid = delradid;
    delete from    RW BiasDyn          where radid = delradid;

    delete from    DP Radar          where radid = delradid;
    delete from    RadarLoc          where radid = delradid;

RETURN NULL;

END;

' LANGUAGE 'plpgsql';

```

Once created, this function can be executed through psql as

```
=> select delete_radar('FCX');
```

```
*****
-- Note that the examples below use " $$ " to signify the body of the function.
-- This is a new feature in Version 8.0.
```

```
-- These examples are provided only as guidelines for illustrating
-- PostgreSQL functions. There is no guarantee of any kind that
-- these functions will perform for any particular task. They are
-- released under a BSD License allowing unrestricted copy, modification
-- and use of these functions in any form.
--     License: BSD license, unrestricted use
--     Author: A. Elein Mustain
```

```
-----
--     Name: trimtablecol()
-- Description: runs trim() on that column in that table
--     Input: tablename, columnname
--     Output: none
--     Effect: Trimmed named column in table, no effect on non-text columns
--     Requires: language plpgsql, works on any table
```

```
-----
```

```
CREATE OR REPLACE FUNCTION trimtablecol(text, text )
```

```
RETURNS VOID AS
```

```
$$
```

```
DECLARE
```

```
qry text;
```

```
tab alias for $1;
```

```
col alias for $2;
```

```
BEGIN
```

```
    qry := 'update ' || tab || ' set ' || \
           col || '=trim(' || col || ')';'
```

```
EXECUTE qry;
```

```
RETURN;
```

```
END;
```

```
$$ LANGUAGE 'plpgsql';
```

```
-----
--     Name: bigbro()
-- Description: Trigger function to log changes of user table
```

```

-
    in human readable form
--      Input: TRIGGER
--      Output: TRIGGER
--      Effect: Insert into bigbro table with changes to user table
--      Requires: language plpgsql, requires user table, bigbro table,
--                  triggers on user table
-----
CREATE TABLE users (
    email text,
    who   text );

CREATE TABLE bigbro (
    who   text,
    what  text,
    tab   text,
    wwhen timestamp,
    change text);

CREATE OR REPLACE FUNCTION bigbro ()
RETURNS TRIGGER as
$$
DECLARE
    changes text;
    qry text;

BEGIN
    -- initialization
    changes := '';
    qry := 'insert into bigbro (who, what, tab, whn, change)' || \
        ' values '(USER, ' || quote_literal(TG_OP || '-' || TG_WHEN) \\
        || ', ' || quote_literal(TG_RELNAME) || ', now(), ';

    -- Create human readable change string
    -- For insert build NEW(who=xxx,email=xxx)
    -- For update build OLD(who=xxx,email=xxx),NEW(who=zzz,email=zzz)
    -- If nothing changed, default to 'No changes.'
    IF TG_OP = 'INSERT' THEN
        changes := 'NEW(who=' || NEW.who || ', email=' || NEW.email || ')';
    ELSIF TG_OP = 'UPDATE' THEN
        IF NEW.who <> OLD.who OR NEW.email <> OLD.email THEN
            changes := 'OLD(who=' || OLD.who || ', email=' || OLD.email || '), ' \\
                || 'NEW(who=' || NEW.who || ', email=' || NEW.email || ')';
        END IF;
    END IF;

    IF changes = '' THEN
        changes := 'No Changes.';
    END IF;

    -- add the changes string to the end of the log query and close it
    qry := qry || quote_literal(changes) || ');';

    -- log the changes in table bigbro
    EXECUTE qry ;

    -- allow insert or update to complete
    RETURN NEW;

```

```

END;
$$ LANGUAGE 'plpgsql';

CREATE TRIGGER insbb
BEFORE INSERT ON users
FOR EACH ROW EXECUTE PROCEDURE bigbro();

CREATE TRIGGER upddb
BEFORE UPDATE ON users
FOR EACH ROW EXECUTE PROCEDURE bigbro();

-----
--      Name:
-- Description:
-
--      Input:
--      Output:
--      Effect:
--      Requires: language plpgsql,
-----

CREATE TYPE mmav( mkey text, minval numeric, maxval numeric, avgval float );
CREATE FUNCTION mmav( text, text, text )
RETURNS SETOF mmav AS
$$
DECLARE
    col alias for $1;
    tab alias for $2;
    tkey alias for $3;
    m mmav%ROWTYPE;
    r RECORD;
    bucket double precision;
    counter integer;
    qry text;
BEGIN
    qry := 'SELECT ' || col || ' AS val, ' \
           || tkey || ' AS keyval FROM ' || tab || \
           ' ORDER BY ' || tkey;

    bucket := 0;
    counter := 0;
    m.maxval := 0;
    m.minval := 0;
    m.mkey := NULL;

    FOR r in EXECUTE $qry LOOP
        IF m.mkey IS NULL THEN
            m.mkey = r.keyval;
        ELSIF m.mkey <> r.keyval THEN
            IF counter <> 0 THEN
                m.avgval := bucket/counter;
            ELSE
                m.avgval := NULL;
            END IF;
        RETURN NEXT m;
        bucket := 0;
        counter := 0;
    END IF;

```

```

m.maxval := 0;
m.minval := 0;
m.mkey   := r.keyval;
END IF;

counter := counter + 1;
bucket  := bucket + r.val;
IF m.maxval <= r.val OR m.maxval = 0 THEN
    m.maxval := r.val;
END IF;
IF m.minval >= r.val OR m.minval = 0 THEN
    m.minval := r.val;
END IF;

END LOOP;

IF counter <> 0 THEN
    m.avgval := bucket/counter;
ELSE
    m.avgval := NULL;
END IF;

RETURN m;
$$
LANGUAGE 'plpgsql';

```